Bulletin of Electrical Engineering and Informatics
Vol. 7, No. 2, June 2018, pp. 199~208
ISSN: 2302-9285, DOI: 10.11591/eei.v7i2.818

An Enhanced FPGA Based Asynchronous Microprocessor Design Using VIVADO and ISIM

Archana Rani, Naresh Grover
Faculty of Engineering and Technology, Manav Rachna International University, Faridabad, India

Article Info

Article history:

Received Oct 07, 2018
Revised May 08, 2018
Accepted May 22, 2018

Keywords:

Asynchronous processor
FPGA
Synthesis
VHDL
XST

ABSTRACT

This paper deals with the novel design and implementation of asynchronous microprocessor by using HDL on Vivado tool wherein it has the capability of handling even I-Type, R-Type and Jump instructions with multiplier instruction packet. Moreover, it uses separate memory for instructions and data read-write that can be changed at any time. The complete design has been synthesized and simulated using Vivado. The complete design is targeted on Xilinx Virtex-7 FPGA. This paper more focuses on the use of Vivado Tool for advanced FPGA device. By using Vivado we get enhaced analysis result for better view of properly Route & Placed design.

Corresponding Author:

Archana Rani,
Faculty of Engineering and Technology,
Manav Rachna International University, Faridabad, India.
Email: Archana.bhatia.pec@gmail.com

1. INTRODUCTION

The FPGA have become very intrinsic part of every digital logic design. The FPGAs are extensively used by all design engineers to Application engineer for their Research, Design and Testing purposes. One can found the use of FPGAs in every digital system ranging from security systems to entrainment set-top boxes. As the technologies are changing the use of FPGAs are changing proportionally. As most of the researchers are trying to use of the features of FPGAs for their new designs and research. By using the same concept I decided to design our 32-bit asynchronous Processor by using FPGAs. However we already have surrounded with every type of the processors ranging from 8-bits to 64-bits. But still the study and use of those processors are still challenging. Now- a- days computers are evolving using RISC (Reduced Instruction Set Computer) Architecture replacing stack architecture with the intention to displace the hypothetical, emulated computer by a real one. The choice of an RISC has become more obvious with the increase in size and complexity of modern processors and software. The hardware designer has a substantial amount of freedom for design by making use of FPGA being much more aware of availability of resources and of its limitations than the software developer.

Before commencing the design of an asynchronous processor we have to first focus on the architecture of Asynchronous processor as well as the various steps involved in such designs in terms of the program cycle. This paper presents processor architecture design, its implementation followed by processor instruction set, data path flow for fetching unit, Register type, I-type and load /store type instruction flow. Thereafter this paper illustrates the internal architecture of the processor. In the end, results have been shown using implementation and simulation windows. The complete design has been written using VHDL and then simulated and synthesized by Vivado.

2. STUDY OF PROCESSOR ARCHITECTURE [1]

The asynchronous processor internal operation is segmented into five pipeline stages and in each of them the operations of the tasks will be performed in the normal cycle of an instruction, i.e. search of the instruction (identified with the IF block), decoding of The instruction (identified with the ID block), execution of the operation (identified with the EX block), memory access (identified with the MEM block) and storage of the operation result (identified with the WB block) [16]. The first stage is Instruction Fetch that comprises of Instruction Memory, Program Counter, and Instruction Register. In this stage, a program counter will extract the next instruction from a location in program memory. It updates the program counter value with the next instruction location sequentially or the location determined by a branch. The second stage is instruction decoding which comprises of register file and the extender (sign & Zero). This stage determines the values on which the control lines should be set as per the instruction. The third stage is the instruction execution stage, where ALU and necessary parts will come into action. In this stage, the instruction is actually sent to the ALU and branch locations are also calculated, the fourth stage is Memory execution stage for accessing of data from system memory. Finally, in Write back stage the values/data written back to the register(s).

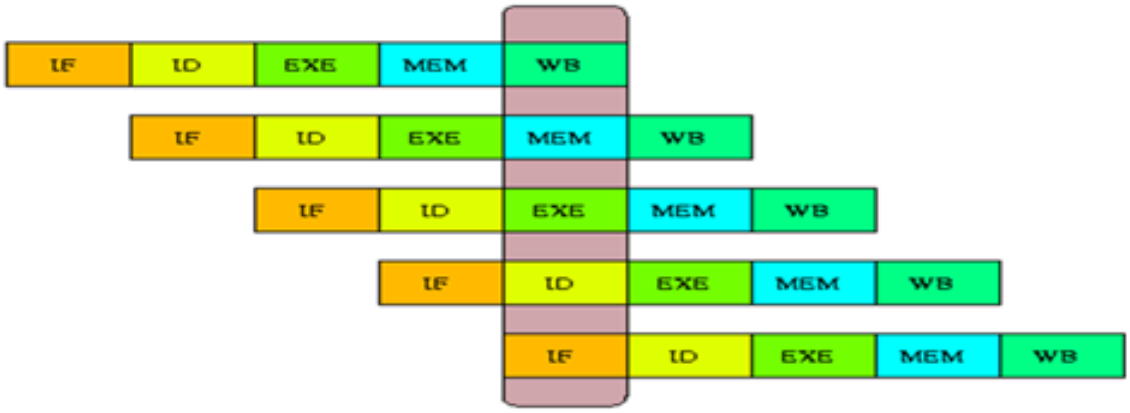

Figure 1. Processing stages

Figure 2 shows the internal architecture of Asynchronous processor, In this processor, fetching the instruction pointed by the Program counter goes to the next unit called decoder which generates the different values of the memory location, as per the instruction fetched from the previous unit. During this control unit has been designed in order to synchronize the various other units such as ALU, data memory or general purpose registers to properly execute the desired instruction. There are other units named as ALU, data memory, and some multiplexers to complete the execution cycle.

The processor has a fetch unit which comprises of PC and ROM section. The processor will come in active state only at the positive edge of clock pulse with the active high reset signal. The Control unit which is in the Decoder part will receive the 32-bit instruction from the address shown by the PC value. The starting address of an instruction is always starts from Zero. The control unit is basically responsible for the synchronization between other units. For this purpose this will generate several other control signals. The order of PC may be modified or randomized by the order of the instruction in our Instruction memory.

The proposed processor is using separate memory for instructions and Data [1]. The capacity of instruction memory i.e. ROM is of 8192*32 in which 8192 are representing the locations where instructions are to be stored with 32-bit data. The structures of instructions are as per ISA (instruction set Architecture). For all stages, there is only one clock cycle needed, while the data memory has the capacity of 64K. Both memories are functioning in falling pulse. The other pulses are used for developing the necessary functions just like pipelining in order to make our processor core faster and much flexible.

The some control signals for these purposes are as follows:

MemRead: if 1, read from memory;
MemWrite: if 1, write to memory;
RegDst: if 1, the destination number for the Write register comes from the Rd field; if 0, it comes from Rt field;

RegWrite: if 1, the general-purpose register selected by the Write register number is written with the value of the Write data input;
AluSrcA: (ALU Source A) if 1, the operand is A register; if 0, the operand is PC;
MemtoReg: if 1, it comes from Memory data register (MDR) and if 0, the value fed to the register file Write data input comes from ALUOut;
IRWrite: if 1, write instruction is performed in IR;
PCWrite: if 1, update the PC;

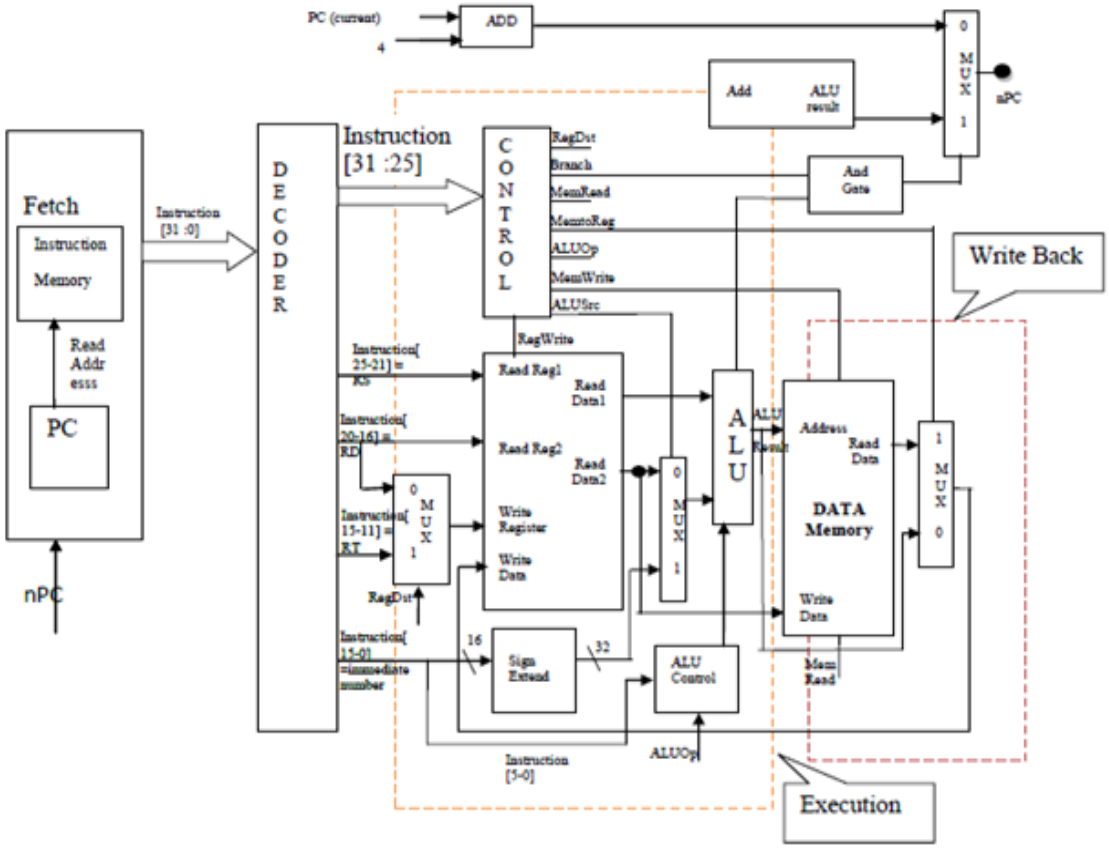

Figure 2. Internal processor architecture [1]

3. VIVADO IDE: AN INTRODUCTION [17]

The Vivado Integrated Design Environment (IDE) provides an intuitive graphical user interface (GUI) with powerful features [Vivado guide]. All of the tools and tool options are written in native Tool Command Language (Tcl) format, which enables the use in both the Vivado IDE or Vivado Design Suite Tcl shell. Analysis and constraint assignment is enabled throughout the entire design process[17]. For example, one can run timing or power estimations after synthesis, placement, or routing. Because the database is accessible through Tcl, changes to constraints, design configuration or tool settings happen in real time, often without forcing re-implementation. One can improve design performance using the new algorithms delivered by the Vivado IDE [17], including:

- Register transfer level (RTL) design in VHDL, Verilog, and SystemVerilog
- Intellectual property (IP) integration for cores
- Behavioral simulation with Vivado simulator
- Vivado synthesis
- Vivado implementation for place and route
- Vivado serial I/O and logic analyzer for debugging
- Vivado power analysis
- SDC-based Xilinx® Design Constraints (XDC) for timing constraints entry
- Static timing analysis

- High-level floorplanning
- Detailed placement and routing modification
- Bitstream generation

The Vivado IDE uses a concept of opening designs in memory. Opening a design loads the design netlist at that particular stage of the design flow, assigns the constraints to the design, and then applies the design to the target device. This provides the ability to visualize and interact with the design at each design stage. You can experiment with different implementation options, refine timing constraints, explore the Vivado IP catalog, perform simulation, and apply physical constraints with floorplanning techniques to help improve design results [17]. Early estimates of resource utilization interconnect delay, power consumption, and routing connectivity can assist with appropriate logic design, device selection, and floorplanning. As the design moves through the implementation flow one can further refine the inputs.

Figure 3 shows the Vivado IDE viewing environment. One can interact with the Vivado IDE through mouse, keyboard, or Tcl input [17].

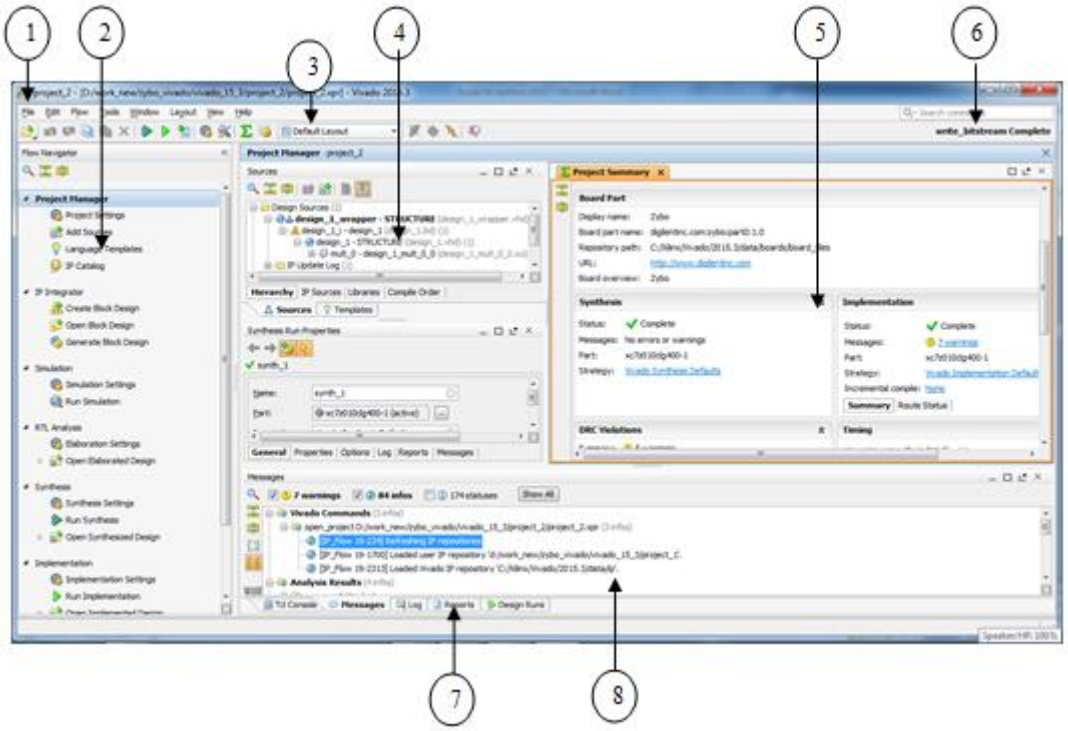

Figure 3. Vivado IDE viewing environment

The main components of the viewing environment are:

1. Menu Bar
2. Flow Navigator
3. Layout Selector
4. Data Windows Area
5. Workspace
6. Project Status Bar
7. Status Bar
8. Results Windows Area

The complete processor has been implemented on the VIVADO tool. All the coding has been done in VHDL and Synthesized by Vivado synthesis tool. Although simulation is the same process and same Layout as of Xilinx ISIM so I used only ISIM simulation window as it has a features of Memory windows. Figure 3 shows the project settings i.e. the target family Virtex-7, with device name XC7Vx485 with 1761 pin packaging. The whole processor code is written using VHDL [1].

4. SYNTHESIS RESULTS

The synthesis and implementation has done by Vivado tool. The process of synthesis generates the number of resources used by our design. The Figure 4 depicts the resource utilization in tabulation and graph format respectively.

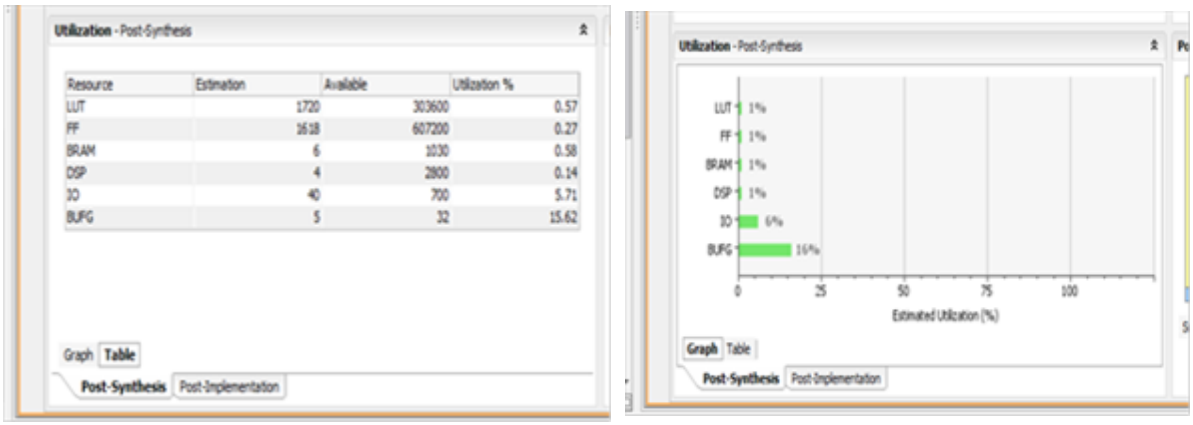

Figure 4. Device utilization using Vivado (post synthesis)

Figure 5 shows the tabulation and graph for the exact device utilization after the design has completely Translate, Map and Place in the target FPGA device(s). The result of this device utilization may vary from device to device selection.

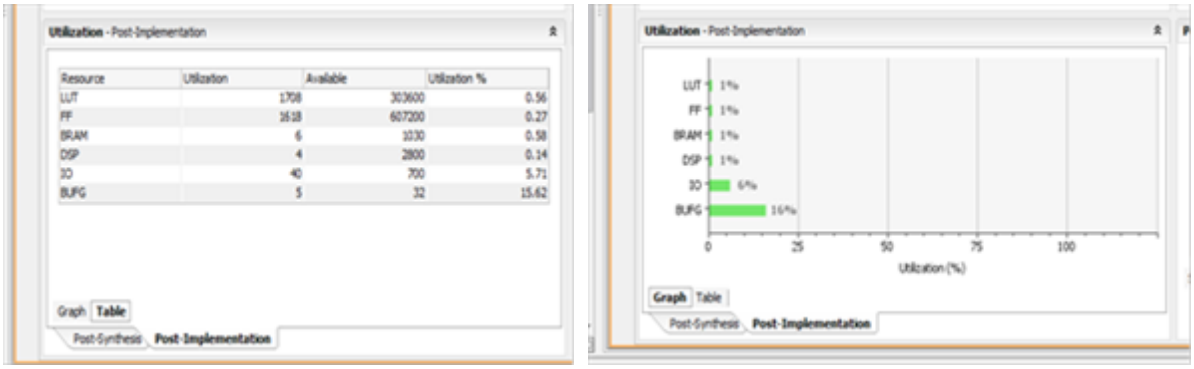

Figure 5. Device utilization using Vivado (post-implementation)

Figure 6 depicts the RTL schematic of our designed IC and their interconnection inside the complete IC development process. The white blocks are our Sub modules interconnected by the green wires/signals. However the connection diagram and the view of internal RTL may be vary as per the design written logic(s) and sometime(s) with the variation of tools also.

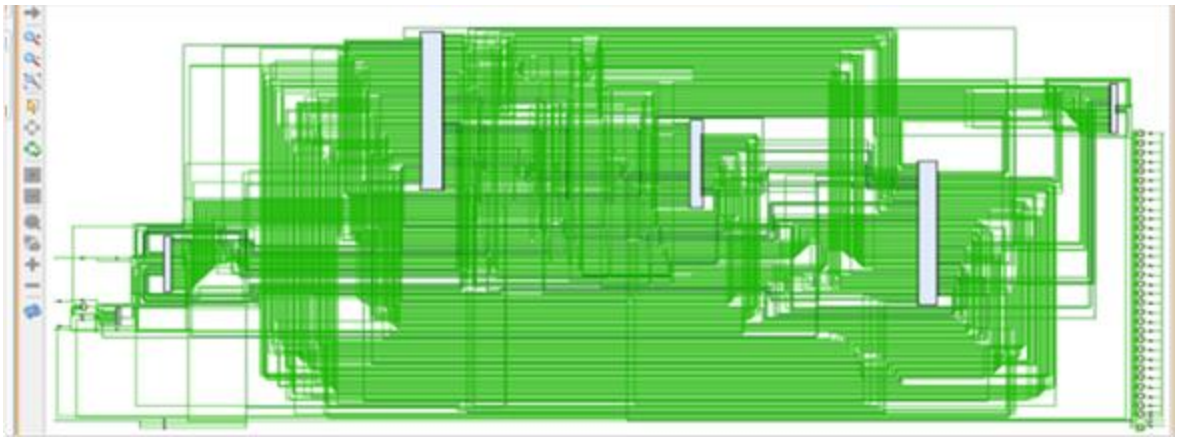

Figure 6. RTL schematic of processor

Figure 7 shows the device implementation in a virtex 7 target device. The design implementation shows our design logic is mapped and placed inside the target FPGA device. In the figure this is highlighted by Green color. There are also some fine green color lines denoting for the IOBs routing connection. The interconnection and placing of design inside a device may also vary from synthesis tool and target device orientation.

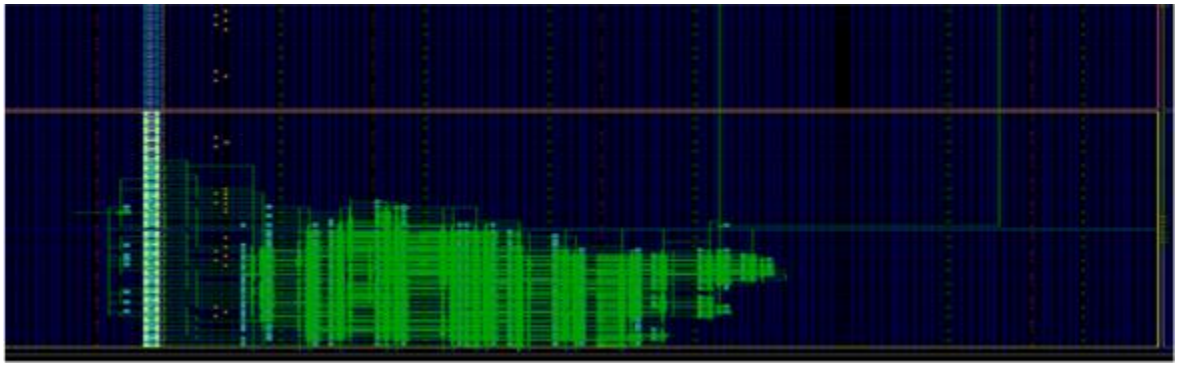

Figure 7. Implemented design of processor in a virtex 7

4.1. Simulation Results

The simulation is used for the logic verification. Here I used ISIM and Vivado simulation tools both. But due to memory windows can be easily shown by using ISIM so I placed the simulation results from ISIM tool. Figure 8 shows the various instructions written into internal memory. From which PC fetches the instruction to be executed. In Figure 4 the table shows the complete structure of the mnemonics for asynchronous processor instruction. The order of instruction execution can be changed as per the designers or consumer requirement. Although once written in Instruction memory the order of the execution of instruction will be as per the order of the mnemonics appears. The Instruction memory is also known as code memory /ROM where all the instructions are going to be stored permanently. The overall execution of any instruction is always been started by the code/instruction memory.

Figure 9 shows the internal register bank arrangements of the designed processor. By default, all the values in registers are (in 32-bit Binary) "00000000000000000000000000000000". We place some random values for initialization in order to get non-zero values. There are thirty-two, 32-Bit registers to temporary hold the data. These registers can be used in any type of addressing modes i.e. Immediate, Register Direct or Indirect addressing modes. The changes in their values will occur automatically when the instruction being fetched and executed. For better explanation we kept the different destination address so that anyone can get found where the changes exactly been.

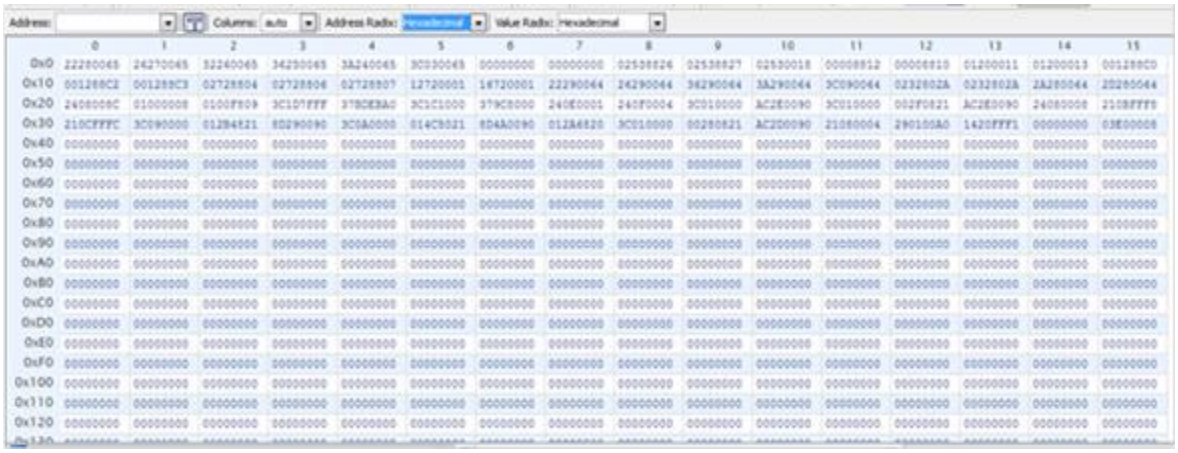

Figure 8. Instruction memory window

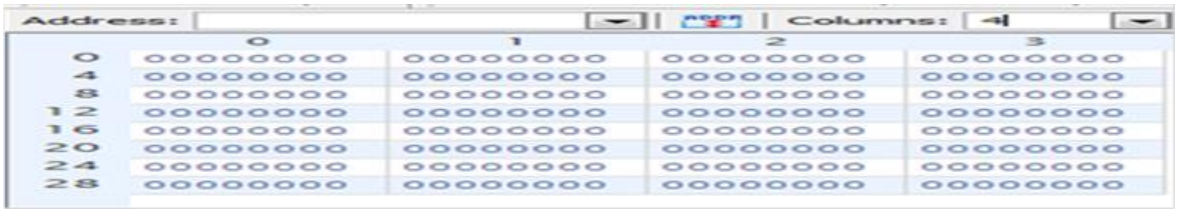

Figure 9. 32, 32-bit register bank (all values are in hex)

Figure 10 shows the result of the addition of Immediate value with another register value and stored into destination register. The whole explanation is carried out by Table 1.

Table 1. ADDI Description

Description:	It adding a register value and a sign-extended immediate value and then stores the result in a register
Operation:	$t = $s + imm; advance_pc (4);
Syntax:	addi $s, $t, imm
Encoding:	001000 sssss ttttt iiii iiii iiii iiii
Decoded	001000 10001 01000 0000 0000 0110 0101

By the simulation diagram it clearly shows that our destination register location is 8 and source location 11h and initially we place some random number in source location i.e. “01100110” in Hex. And after addition as shown in figure above register 8 has loaded with value “01100166” after added by “65h”.

This simulation window in Figure 11 shows the result of ANDing of an immediate number with value of any source register location. Here the immediate number is same i.e. “65h” and the source location is “11h”. After AND operation the destination location “06h”is updated by the value “00000001h”. The below figure shows the updated value at location “06h”, the “11h” having the “01110001h” and immediate number was “0000000001100101b” or “65h”. The result is “00000001h”. Table 2 shown bitwise and immediate AND operation.

Table 2. Bitwise and Immediate AND Operation

Description:	Performing Bitwise ANDing with a register and an immediate value
Operation:	$t=$s & imm; advance_pc (4);
Syntax:	andi $s, $t, imm
Encoding:	001100 sssss ttttt iiii iiii iiii iiii
Decoded	001100 10001 00110 0000000001100101

Figure 11. AND immediate and updated status value in register bank

The above Figure 12 shows the result of the Or immediate number with the specified location i.e. “11h” whose value is “01110001h” with immediate number “65h”. the result “01110065h” is updated in the destination register location i.e. 05h.

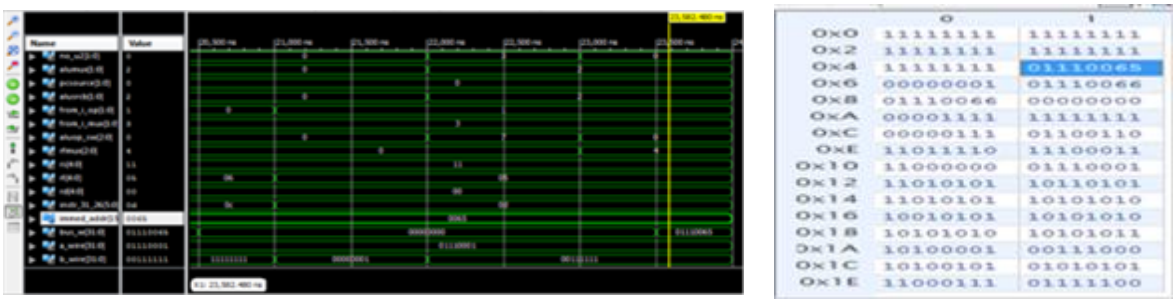

Figure 12. OR immediate and updated status result in register bank

4.2. Overall Processor Simulation Results

Figure(s) 13 and 14 are showing the overall processor results. While initializing the processor with rst =’1’. Our processor starts fetching the instruction from ROM/ Instruction memory which contains the Store, Load word and Addition and Subtraction operation. Due to large simulation signals we had divided our simulation windows in two halves. The first half i.e. figure 19 shows the signals like clk, rst, Aluop, Alusw, AlusrcA, IRwrite etc. These signals are basically coming from the control Unit to bind the overall instruction execution. For simplicity we had taken out Bus_r(31:0) for the final output. Although we can verify these data into the internal memory location(s) or Register Memory Window.

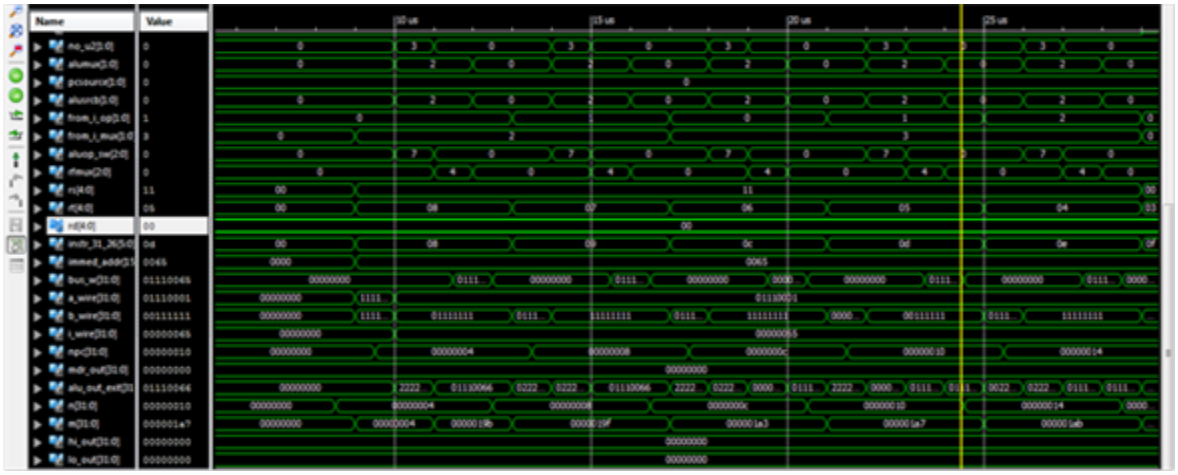

Figure 13. Overall processor simulation result (1)

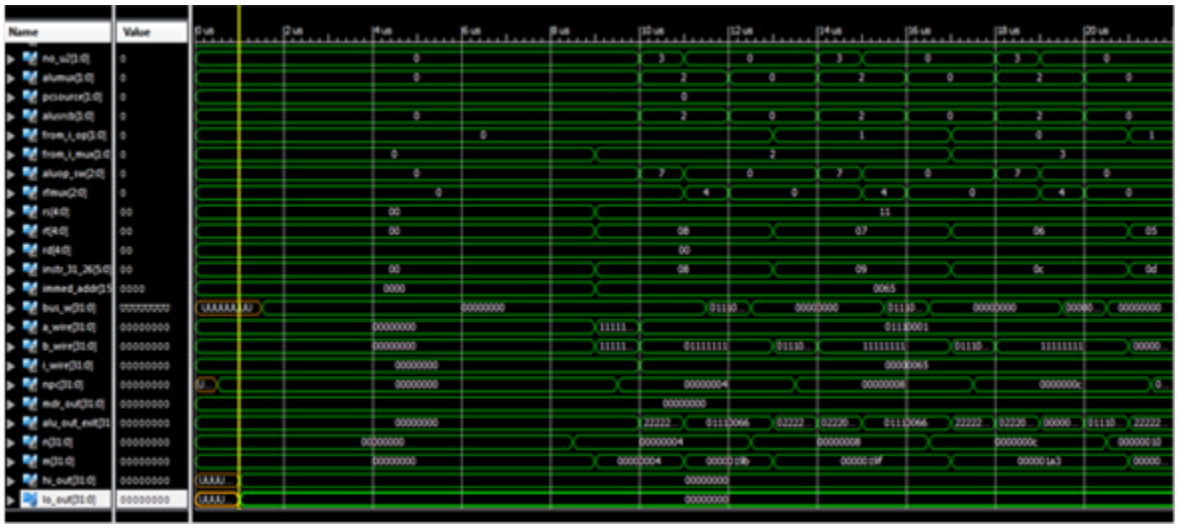

Figure 14. Overall processor simulation result (2)

5. CONCLUSION

The **32**-bit fully functional asynchronous processor has been designed using VHDL. A fully asynchronous processor has been implemented that is comprised of five stages. The work can be potentially improved by reducing I and R-type instructions. The functional simulation shows that proposed processor executes all the various instructions efficiently. The proposed design being an open core is more advantageous as compared to the existing commercial microprocessor for better understanding of internals of the asynchronous microprocessor. The complete core has been checked and verified by the Vivado. As the Vivado is the new tool for the advance level of FPGAs architecture(s) such as Virtex-7, Kintex-7 and Zynq-7. In the results we can easily find out that the use of Vivado is significantly far better experience for getting any type of results regarding our design process. Vivado make comfortable as we get our all types of results in the single go. The design of this 32-bit asynchronous microprocessor implemented using FPGA can be further optimized to reduce its power and area by using any of the following optimization techniques Technology mapping & Logic Optimization.

REFERENCES

[1] Archana Rani, Dr. Naresh Grover,"Novel Design of 32 bit asynchronous(RISC) micfroprocessor & its implementation using FPGA"I.J.Information Engineering and electronic Business,2018,1,pp39-47.
[2] Gang Qin, Yanii Ma, Xiangzhouzang, Ming Zhang,"Design of fuzzy adaptive PID temperature Controller based on FPGA",Telkomnika Indonesian Journal Of Electrical Engg.vol11No.10,2013pages6008-6016.33.
[3] NaWu,Yinjing Guo,YongginWie,ShuxianFan,"Design of New Transformer Protection Device based on Wavelet Energy entropy neural Network Theory & FPGA",Telkomnika Indonesian Journal of Electrical Engg.vol11,2013pages 6746-6754.
[4] Aneesh, R.; Jiju, K. "Design of FPGA based 8-bit RISC controller IP core using VHDL", India Conference (INDICON), 2012 Annual IEEE, On page(s): 427 – 432
[5] Anjana R & Krunal Gandhi, "VHDL Implementation of a MIPS RISC Processor", August 2012, International Journal of Advanced Research in Computer Science and Software Engineering, pp 83-88
[6] Bhosle Preetam, Hari Krishna Moorthy, "FPGA Implementation of Low Power Pipelined 32-bit RlSC Processor", Proceedings of International Journal of Innovative Technology and Exploring Engineering (IJITEE), ISSN: 2278-3075, Vol-I, Issue-3, August 2012.
[7] Ferdous, T. "Design and FPGA-based implementation of a high performance 32-bit DSP processor", Computer and Information Technology (ICCIT), 2012 15th International Conference, on page(s): 484 – 489
[8] M.Indu,Kumar Arun, "Design of Low Power Pipelined RISC Processor", International Journal of Advanced Research in Electrical & electronics & instrumentation Engineering, vol.2, no.3, pp.3747-3756 , August 2013.
[9] MD.Shabeena Begum, M.Kishore Kumar, "FPGA based implementation of 32 bit risc processor", International Journal of Engineering Research and Applications (IJERA), pp 1148-1151
[10] Mohammad Imran, Ramananjaneyulu, "FPGA Implementation of a 64-bit RlSC Processor Using VHDL", Proceedings of International Journal of Reconfigurable and Embedded Systems (IJRES),ISSN:2089-4864,Vol-l, No.2, July 2012.

[11] Purna Addanki Ramesh,Ch.Pradeep,"FPGA Based Implementation of Double Precision Floating point Adder/Subtarctor Using Verilog", Proceedings of International Journal of Emerging Technology and Advanced EngineeringISSN-2250-2459,Vol-2,lssue 7,July 2012.
[12] Uma R., "Design and Performance analysis of 8 bit RISC Processor Using Xilinx Tool", International Journal of Engineering Research and Application, vol.2, no.2,pp. 53-58, April 2012.
[13] Ritpurkar Sagar P., Prof. Mangesh N. Thakare, Prof. Girish D. Korde ," Review on 32-bit MIPS RISC Processor using VHDL", IOSR Journal of Electrical and Electronics Engineering (IOSR-JEEE), PP 46-50
[14] Wikipedia https://en.wikipedia.org/wiki/Asynchronous_circuit
[15] http://www.alteraforum.com/forum/forum.php
[16] https://en.wikipedia.org/wiki/MIPS_architecture
[17] Vivado Design Suite User Guide: Design Flows Overview (UG892)

BIOGRAPHIES OF AUTHORS

Archana Rani Bhatia is a PhD. Scholar from Manav Rachna International University, Faridabad. Had completed Post Graduation in Electronics Product Design and Technology from Punjab Engineering College Chandigarh in 2008 and did Graduation in Electronics and Communication Engineering, with sound working experience over 6.6 years in various electronic works, related to Teaching, Research & Industry side.

Naresh Grover did his B.Sc (Engg.) in 1984 and M.Tech in Electronics and Communication Engineering in 1998 from REC Kurukshetra (Now NIT Kurukshetra). He has a rich experience of 33 years in academics. He has authored two books on Microprocessors and is a co-author for a book on Electronic Components and Materials. His core area of interest is Microprocessors and Digital System Design. Presently he is Dean-Academics at Manav Rachna International University, Faridabad.